How Much CPU Time?

Expressing Meaningful Processing Requirements among Heterogeneous Nodes in an Active Network

Virginie Galtier, Stefan Leigh, Kevin Mills, Doug Montgomery,
Mudumbai Ranganathan, Andrew Rukhin, Debra Tang

NIST

Outline of Talk

- The Problem Statement
- The Main Sources of Variability Affecting CPU Time Requirements of an Active Application
- Modeling Active Network Nodes and Active Applications
  - Active Network Node Model
  - Active Application Model
  - Active Application Model Transforms
- Calibrating Active Network Nodes
- Generating Active Application Models
- Proof-of-Concept Results
- Potential Benefits of Success
- Future Work

The Problem Statement

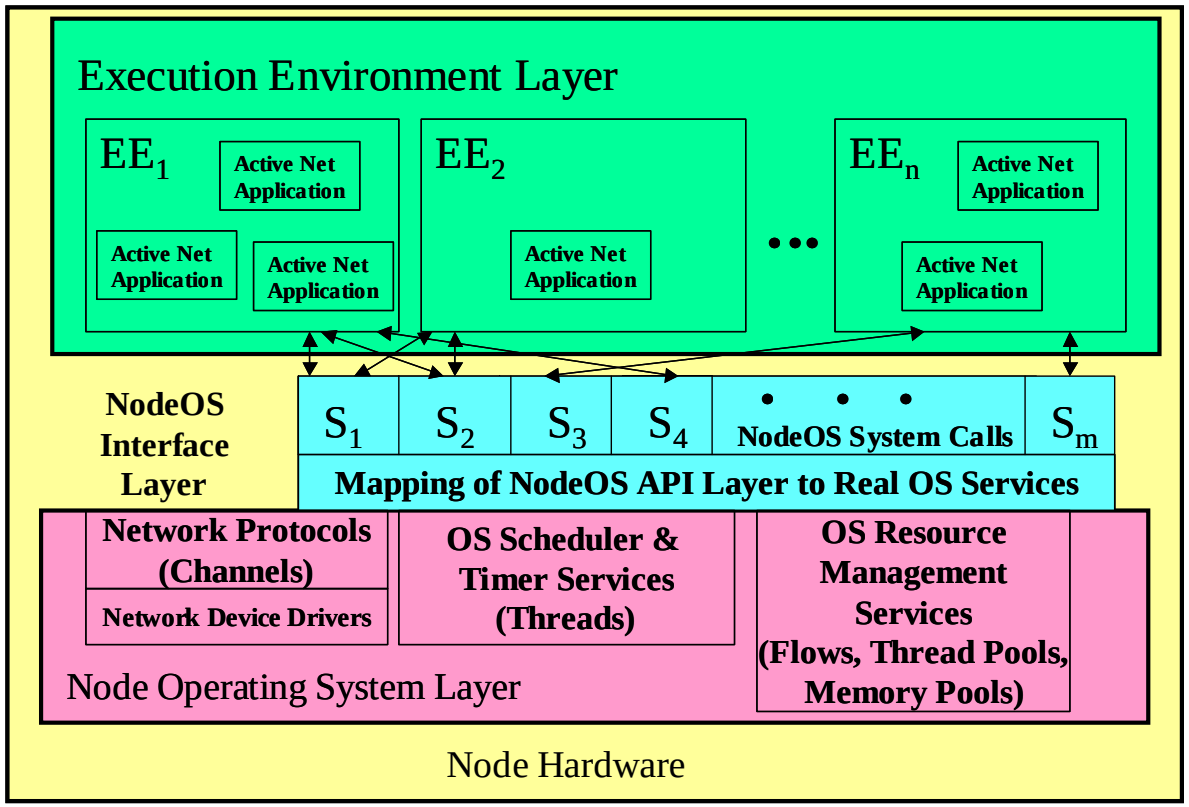
How can one express the CPU time requirements of an Active Application in a form that can be meaningfully interpreted among heterogeneous nodes in an Active Network?

- In current network switches and routers, well-known, system-independent metrics exist for two resources: bandwidth (bits per second) and memory (bytes or byte/seconds). What about CPU cycles?

- Currently, per-packet processing requirements in a network node are fairly homogeneous – Active Networks will change that situation.

- So an accepted, system-independent means of expressing CPU time requirements will be needed to enable allocation and management of CPU cycles among active network nodes.

A Conceptual Model of an Active Node



Three Layer Model

- **Execution Environment**
- **Node OS Interface**
- **Node Operating System**

Plus the Node Hardware

Helps to identify the sources of variability affecting CPU time requirements in an Active Application.

NIST

Five Main Sources of Variability

Any effective metric for CPU time usage in an Active Network Node must account for five main sources of variability:

1. Raw Performance of Node Hardware
2. Specific EE in which AA executes, along with the mapping of the EE virtual machine to the node hardware
3. Mapping of Node OS system calls to real system calls in the host operating system
4. Implementation of real system calls within the host operating system, including the selection of specific protocol modules to implement each instance of a Node OS channel
5. Behavior of the AA itself

Proposed Three-Part Model for Active Network Nodes and Active Applications

- Active Network Node Model
  (accounts for first four sources of variability)
- Active Application Model
  (accounts for fifth source of variability)
- Active Application Model Transforms

Each of these is explained in the next few slides.

Active Network Node Model

Example for Two Nodes (Node A and B)

Node A: Execution Environment Vectors

	EE_1	EE_2	EE_3
$EE_{reference}$	.456 s	.758 s	.326 s
EE_{nodeA}	.228 s	.378 s	.175 s
$EE_{reference}/EE_{nodeA}$	2	2.005291	1.86285714

Node A: Node OS Call Vectors

	S_1	S_2	S_3	S_4
$S_{reference}$	.0054 ms	.0109 ms	.0012 ms	.0075 ms
S_{nodeA}	.0108 ms	.0179 ms	.0036 ms	.0167 ms
$S_{reference}/S_{nodeA}$	.5	.61	.33	.45

Node B: Execution Environment Vectors

	EE_1	EE_2	EE_3
$EE_{reference}$	.456 s	.758 s	.326 s
EE_{nodeB}	.052 s	.084 s	.033 s
$EE_{reference}/EE_{nodeB}$	8.77	9.02	9.88

Node B: Node OS Call Vectors

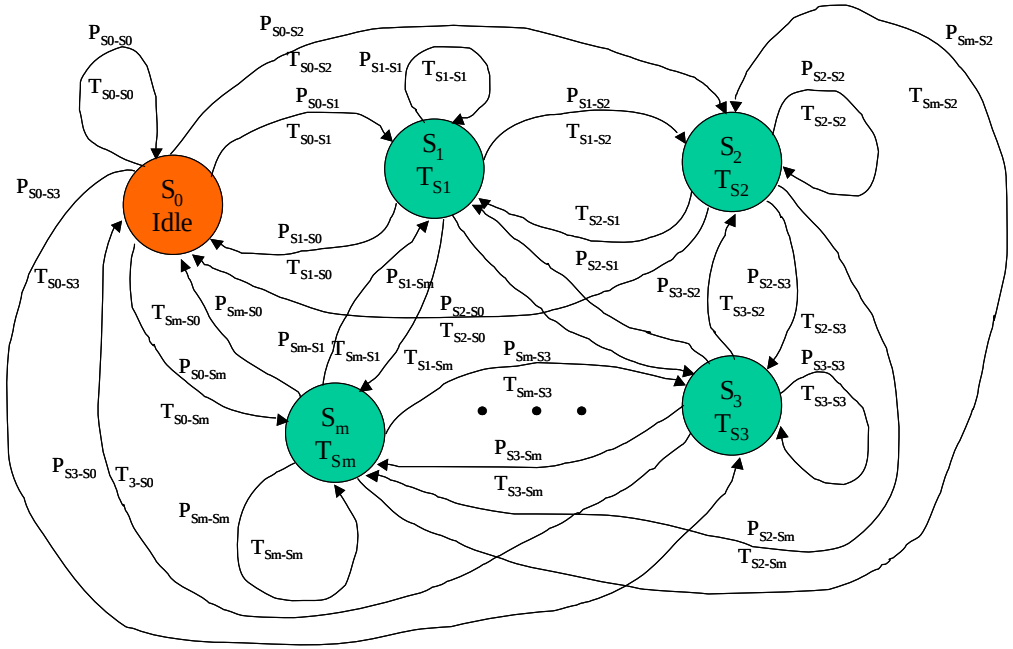
	S_1	S_2	S_3	S_4
$S_{reference}$	.0054 ms	.0109 ms	.0012 ms	.0075 ms
S_{nodeB}	.0045 ms	.0099 ms	.0009 ms	.0069 ms
$S_{reference}/S_{nodeB}$	1.2	1.1	1.33	1.09

- Performance of the local node and of a reference node with respect to a benchmark workload for each EE
- Ratio of reference node to local node performance for each EE benchmark
- Performance of the local node and of a reference node with respect to a benchmark workload for each Node OS call
- Ratio of reference node to local node performance for each Node OS call

Node B is about 5x faster

Active Application Model

Semi-Markov Model



Semi-Markov Model chosen as a first coarse approximation of a more complex reality. Measurement data will tell the real story – leading us to revise this model as necessary.

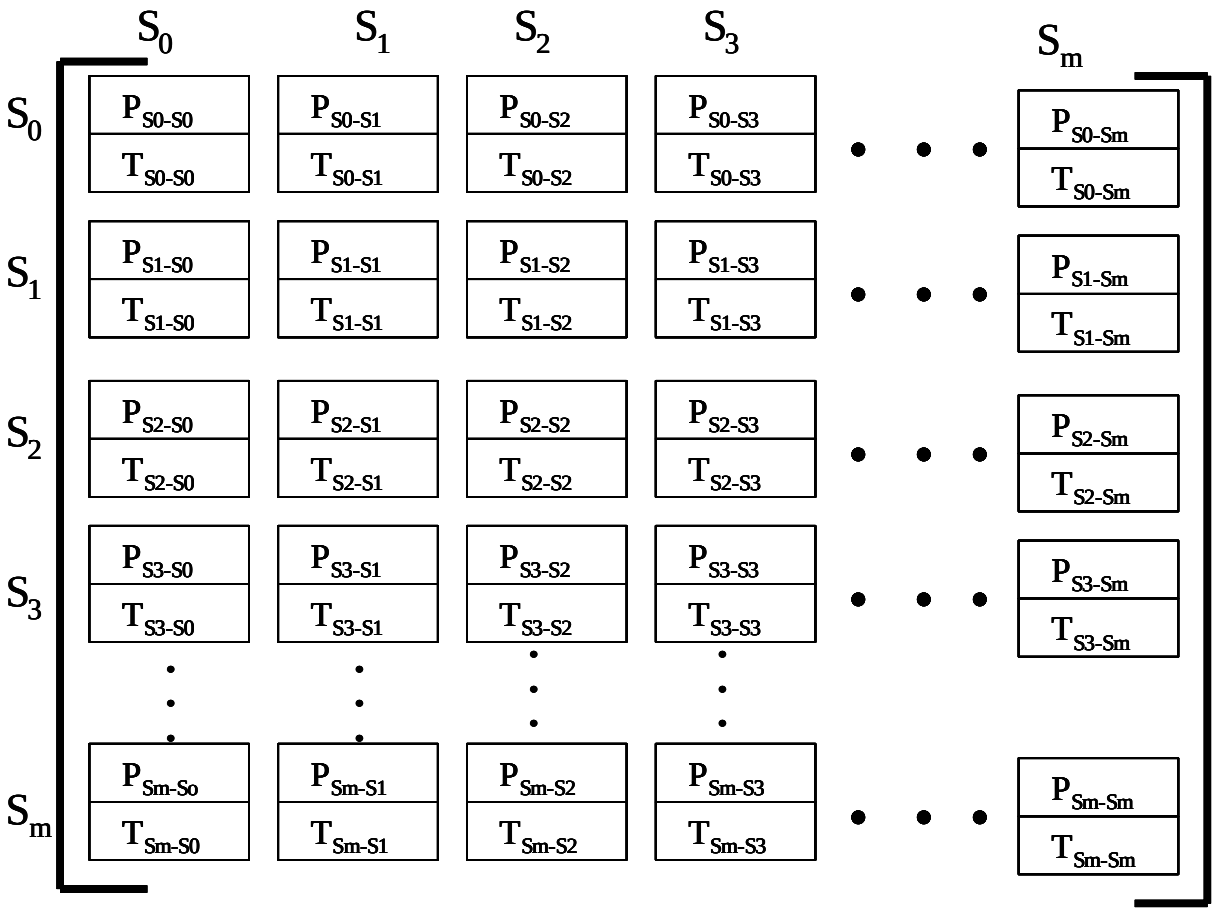
- States denote AA calls to Node OS
- Transitions denote AA execution within EE
- Idle (Red) state denotes beginning and end of an AA execution thread
- T's are CPU times attached to states and transitions
- P's are probabilities of each transition

Active Application Model (in Matrix Form)

Node OS Call State Vector

System Call	CPU Time
S_0	Idle
S_1	T_{S1}
S_2	T_{S2}
S_3	T_{S3}
⋮	⋮
S_m	T_{Sm}

Execution Environment Transition Matrix



How might a Node OS use an Active Application Model?

- We can pool all states in the AA model beyond Idle (S0) into one composite state (SA), creating a two-state Markov chain.
- If we assume that this chain is stationary, then the distribution of measured dwell time in each state will be exponential.
- Given this assumption, the time to leave state S0 and SA can be written:

$P(\varepsilon > t) = e^{**}(-\lambda^0 t)$ (Time to leave S0)

$P(T > t) = e^{**}(-\lambda t)$ (Time to leave SA)

The distribution of average dwell times in SA is $\lambda^* = 1/T$, **where T can be computed as:**

$$\frac{\left(\sum_{j=1}^{n} Xj\right)}{n}$$

where each Xj represents the observed average dwell time in one of the component states aggregated together to form SA.

The distribution of average dwell times can be partitioned into two parts α and $1-\alpha$, where each partition represents a region in which some proportion of the dwell times in SA fall. Since for an exponential distribution $\alpha = e^{-\lambda^* t_\alpha}$, t_α denotes a value above which α percent of the observations will be found. t_α can be found as follows.

$$\lambda^* t_\alpha = -\log \alpha$$

$$t_\alpha = -(1/\lambda^*) \log \alpha$$

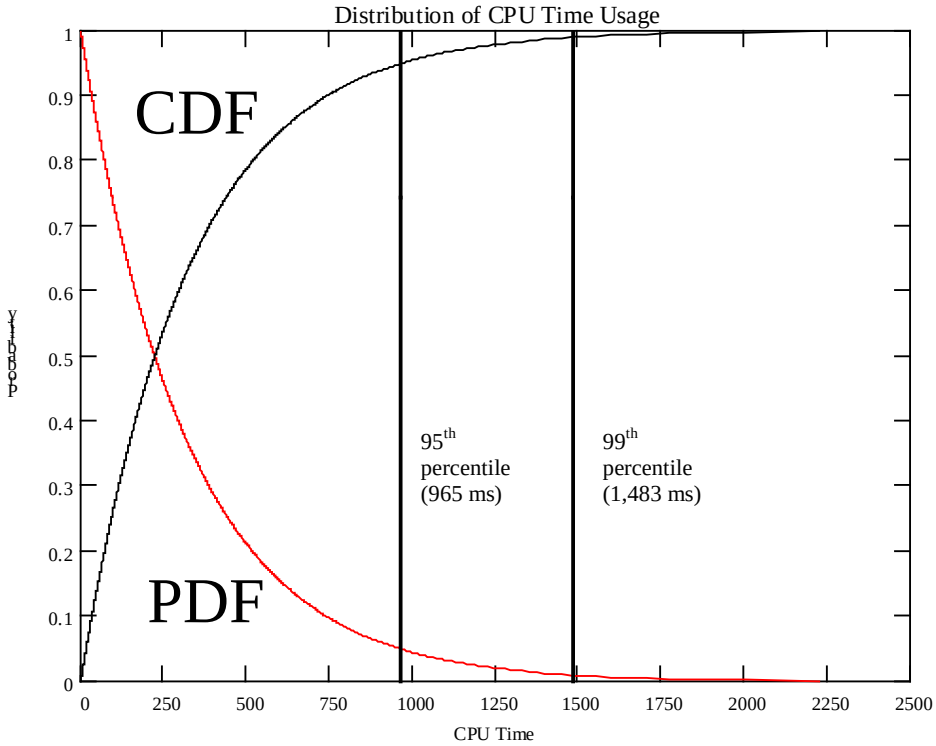
Substituting 1/T for λ^*: $t_\alpha = -T \log \alpha$

This equation leads to an easily computable threshold value.

Example: Consider the Following Model for an AA Executing on an Active Network Node A

AA Model for Node A

Node OS Call State Vector			Execution Environment Transition Matrix				
System Call	CPU Time		To S_0	To S_1	To S_2	To S_3	To S_4
S_0	0.0000	From S_0	0|0	.8|1234	.2|457	0|0	0|0
S_1	0.0114	From S_1	.05|2345	.6|347	.25|423	.1|256	0|0
S_2	0.0165	From S_2	.25|337	.15|1115	.2|313	.2|109	.2|92
S_3	0.0280	From S_3	.01|1632	.55|756	.04|577	.3|188	.1|89



Cumulative Distribution Function and Probability Density Function computed from AA Model above using equations covered on preceding slide. ($\lambda* = 0.0031$)

The expected CPU time to execute the AA on Node A is ~322 ms, while 95 percent of all executions should require <= 965 ms of CPU time and 99 percent of all executions should require <= 1.483 seconds of CPU time.

Application Model Transforms: Node-to-Reference (NR) Transform

Node to Reference Transformation

n := the index for the specific execution environment used for the application
m := the number of system calls supported by a NodeOS

for i from 0 to m
$SC_{vector}[i] := S_{reference}[i]/S_{node}[i] * SC_{vector}[i]$
for j from 0 to m
$EE_{matrix}[i,j].T := EE_{reference}[n]/EE_{node}[n] * EE_{matrix}[i,j].T$
end for
end for

AA Model for Reference Node and for Transmission on the Network

Node OS Call State Vector			Execution Environment Transition Matrix				
System Call	CPU Time		To S_0	To S_1	To S_2	To S_3	To S_4
S_0	0.0000	From S_0	0|0	.8|2468	.2|914	0|0	0|0
S_1	0.0057	From S_1	.05|4690	.6|694	.25|846	.1|512	0|0
S_2	0.0101	From S_2	.25|674	.15|2230	.2|626	.2|218	.2|184
S_3	0.0092	From S_3	.01|3264	.55|1512	.04|1154	.3|376	.1|178
S_4	0.0071	From S_4	.6|7042	.1|1964	.2|690	.05|690	.05|214

Application Model Transforms: Reference-to-Node (RN) Transform

Reference to Node Transformation

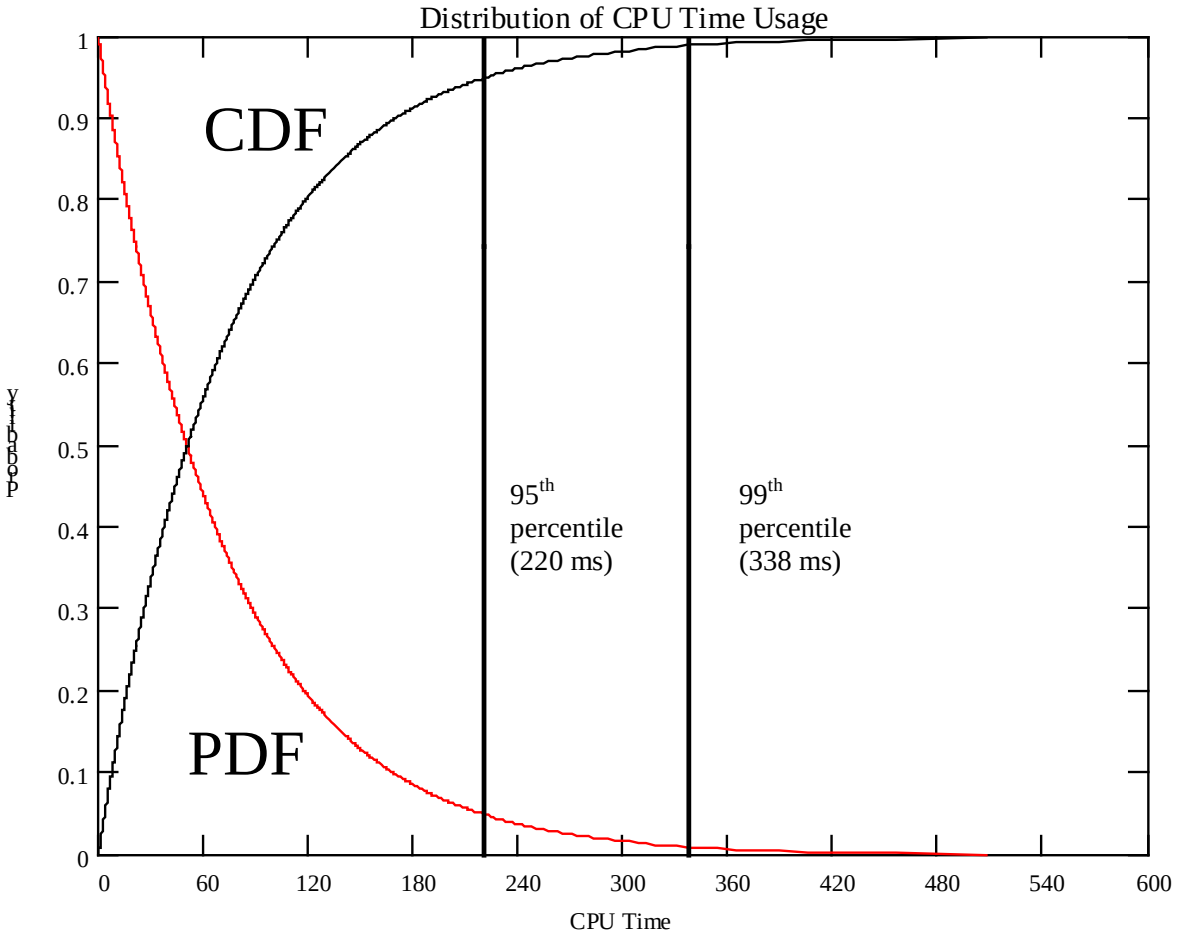
n := the index for the specific execution environment used for the application
m := the number of system calls supported by a NodeOS

for i from 0 to m
$SC_{vector}[i] := S_{node}[i]/S_{reference}[i] * SC_{vector}[i]$
for j from 0 to m
$EE_{matrix}[i,j].T := EE_{node}[n]/EE_{reference}[n] * EE_{matrix}[i,j].T$
end for
end for

AA Model for Node B

Node OS Call State Vector			Execution Environment Transition Matrix				
System Call	CPU Time		To S_0	To S_1	To S_2	To S_3	To S_4
S_0	0.0000	From S_0	0|0	.8|281	.2|104	0|0	0|0
S_1	0.0047	From S_1	.05|535	.6|79	.25|96	.1|58	0|0
S_2	0.0092	From S_2	.25|77	.15|254	.2|71	.2|25	.2|21
S_3	0.0069	From S_3	.01|372	.55|172	.04|132	.3|43	.1|20
S_4	0.0065	From S_4	.6|803	.1|224	.2|79	.05|79	.05|24

Recalculating CPU Time Requirements for the AA on Active Network Node B



Cumulative Distribution Function and Probability Density Function computed from AA Model using figures from the preceding slide. ($\lambda^* = 0.0136$)

The expected CPU time to execute the AA on Node B is ~73 ms, while 95 percent of all executions should require <= 220 ms of CPU time and 99 percent of all executions should require <= 338 ms of CPU time.

The AA executes on Node B about 5x faster than on Node A.

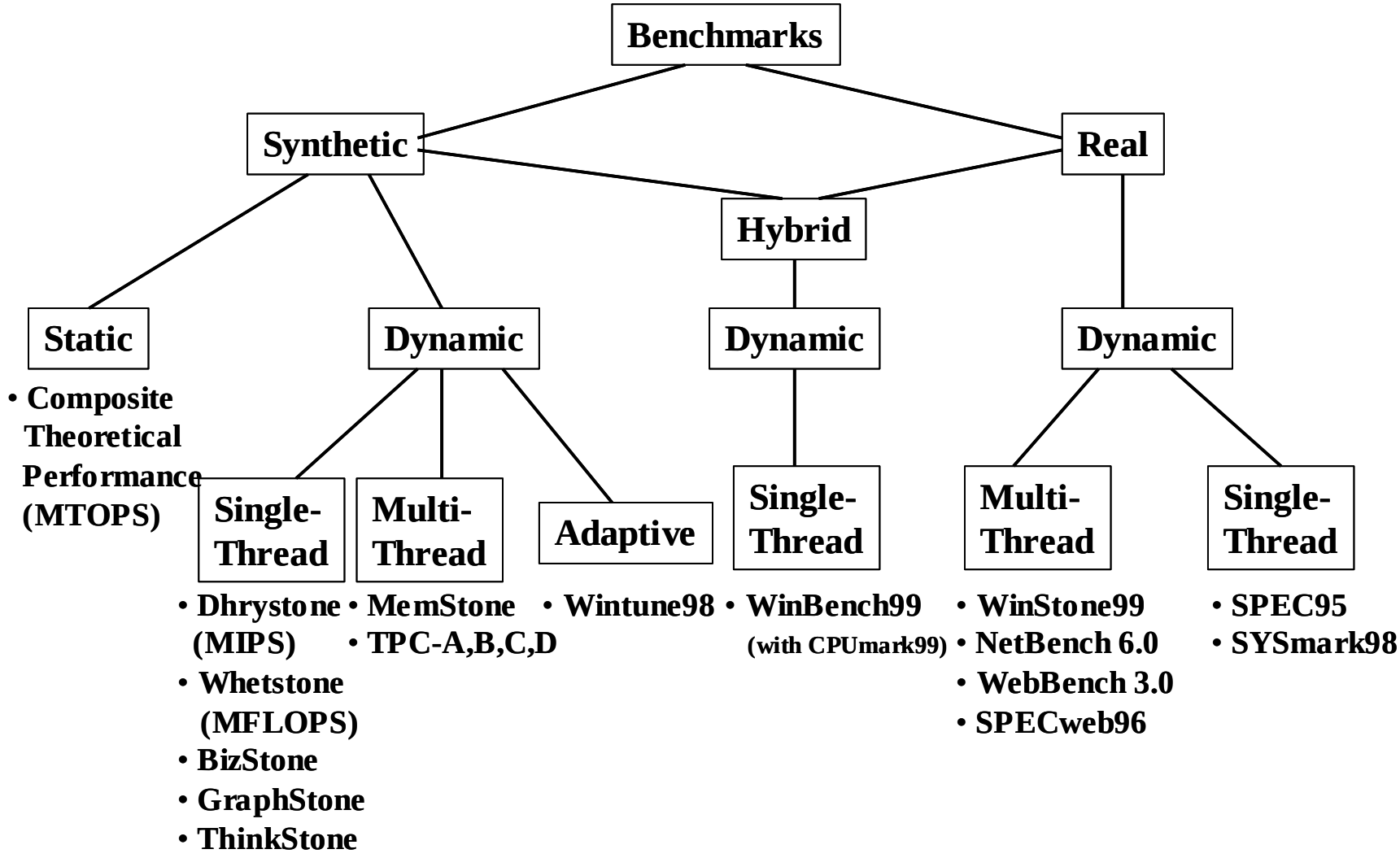
NIST

Determining Models for Active Nodes and Applications

- How can Active Nodes be calibrated?
- How can Active Application models be generated?

Each of these topics is discussed in the next few slides.

A Taxonomy of Selected Existing Computer Performance Benchmarks



Calibrating Active Network Nodes

- **Possible approaches to defining benchmark workloads for Active Nodes**
  - Use **real Active Applications** to construct a workload for each EE
  - Use **representative applications** that behave as we expect major classes of AAs to behave
  - Use a **synthetic benchmark** that repeatedly exercises all functions in an EE
  - Use a **hybrid approach**
  - Benchmark only a reference node and then use a static calculation (e.g., MTOPS) to **estimate performance** on other nodes
- **Node OS call calibration can be done with a synthetic benchmark program that repeatedly exercises each system call**
- **When and how to run the calibration workloads?**
  - **Off-line** (needs no run-time resources, but might lag system configuration changes)
  - **Boot-time** (needs no run-time resources and will catch many configuration changes, but could lengthen the boot process substantially and may not work well with future dynamic operating systems)
  - **Off-line with Run-time Adjustments** (advantages of off-line and should also catch configuration changes with some lag time, but will incur execution overhead and might be difficult to design and implement)

Generating Active Application Models

During testing process an AA is run through many execution paths – an execution trace can be generated in a form similar to the following:

$<SC_i>$ $<SC_j>$ $<SCT_i>$ $<EET_{i,j}>$ $<CPU_{i,j}>$

Where each line represents a transition between two Node OS system calls, and

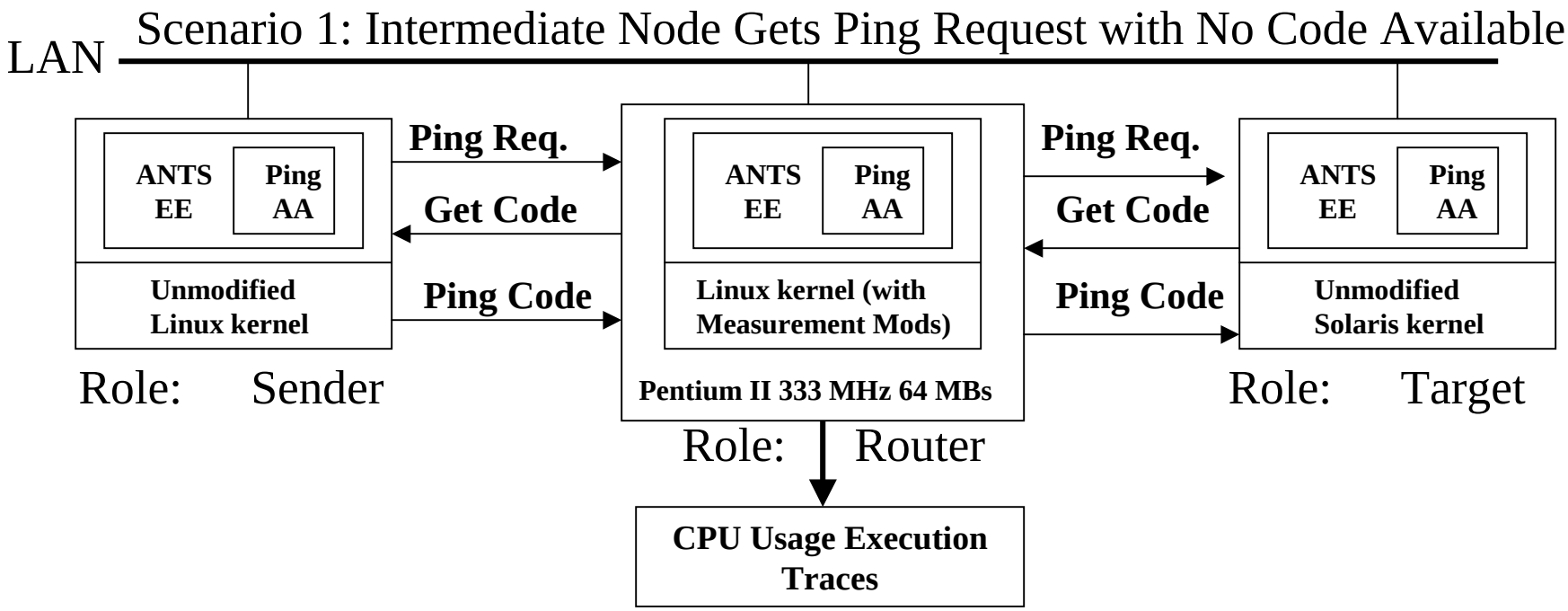
$<SC_i>$ is a unique integer number assigned to identify the "from" NodeOS system call,
$<SC_j>$ is a unique integer number assigned to identify the "to" NodeOS system call,
$<SCT_i>$ is the CPU time spent while executing the "from" system call,
$<EET_{i,j}>$ is the CPU time spent while executing in the EE between $<SC_i>$ and $<SC_j>$, and
$<CPU_{i,j}>$ is total of $<SCT_i>$ + $<EET_{i,j}>$.

A program can be written to automatically generate an AA Model (in vector and matrix form) from such a trace.

Proof-of-Concept: Trace Generation

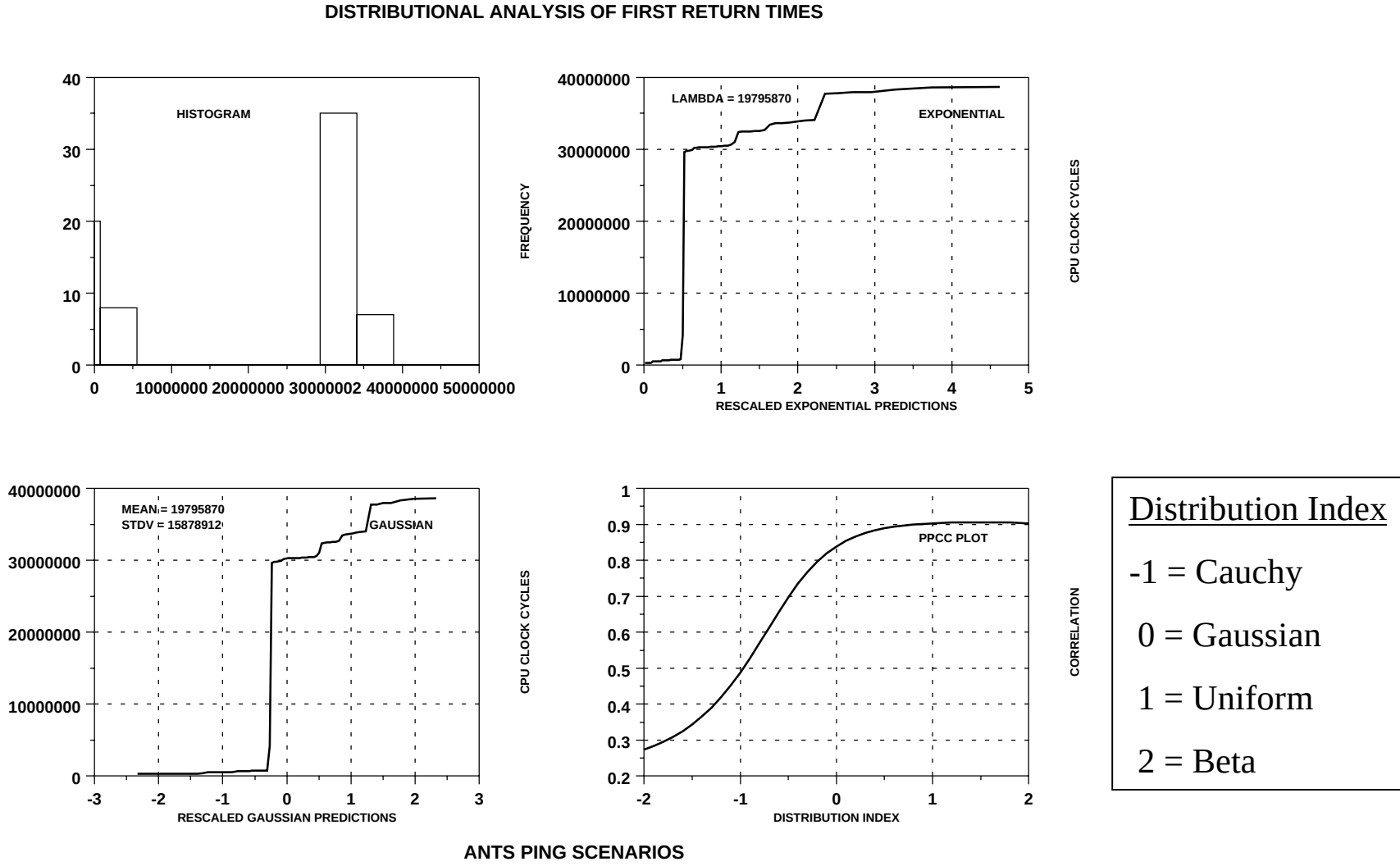
- Modified Linux kernel to generate CPU usage execution traces with minimal measurement overhead
  - Retrieve CPU time used by EE process when entering and exiting each system call, including the scheduler, and write a trace log event
  - Needed to use special Pentium instructions to grab CPU time in nanosecond granularity for measuring system calls
- Generated CPU usage execution traces for several AAs in the ANTS EE running on top of Linux
  - Ping, Auction, Multicast, and TCP denial-of-service defense
  - Ran a number of execution scenarios for each application
- An example follows

Proof-of-Concept: An Example



Scenario 2: Intermediate Node Gets Ping Request with Code Available
Scenario 3: Ping as Source Node with Intermediate Node Needing Code
Scenario 4: Ping as Source Node with Intermediate Node Having Code
Scenario 5: Target of Ping but Needing Code
Scenario 6: Target of Ping but Having Code
. . .
Scenario n: According to Application Test Plan

Proof-of-Concept:Results
Initial First-Return Times for ANTS Ping



Distribution Index

-1 = Cauchy

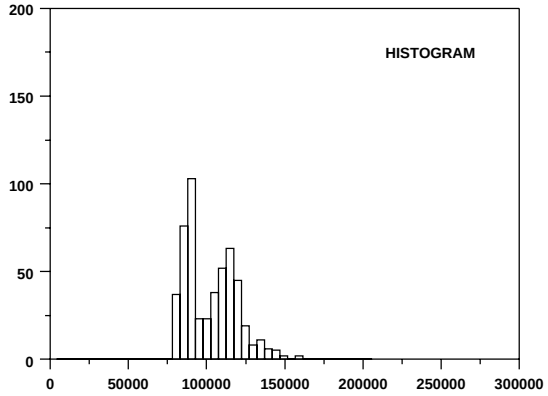
0 = Gaussian

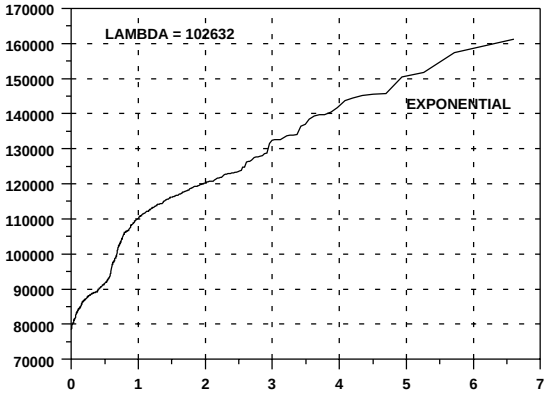
1 = Uniform

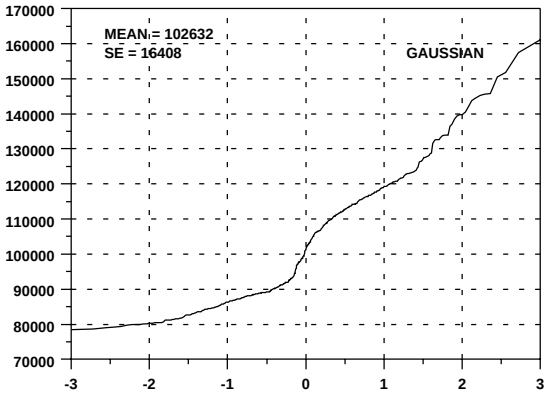
2 = Beta

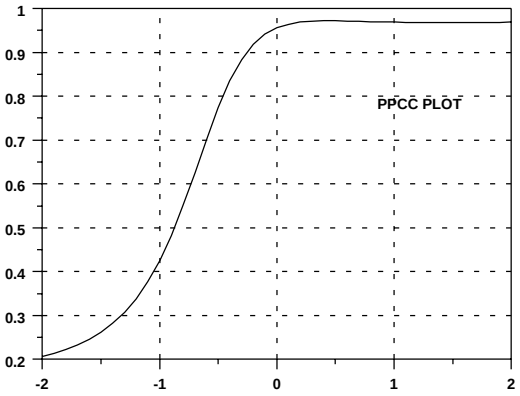
Proof-of-Concept:Results
Transitions between a State Pair for ANTS Ping

DISTRIBUTIONAL ANALYSIS OF DWELL+TRANSITION TIMES









Distribution Index

-1 = Cauchy

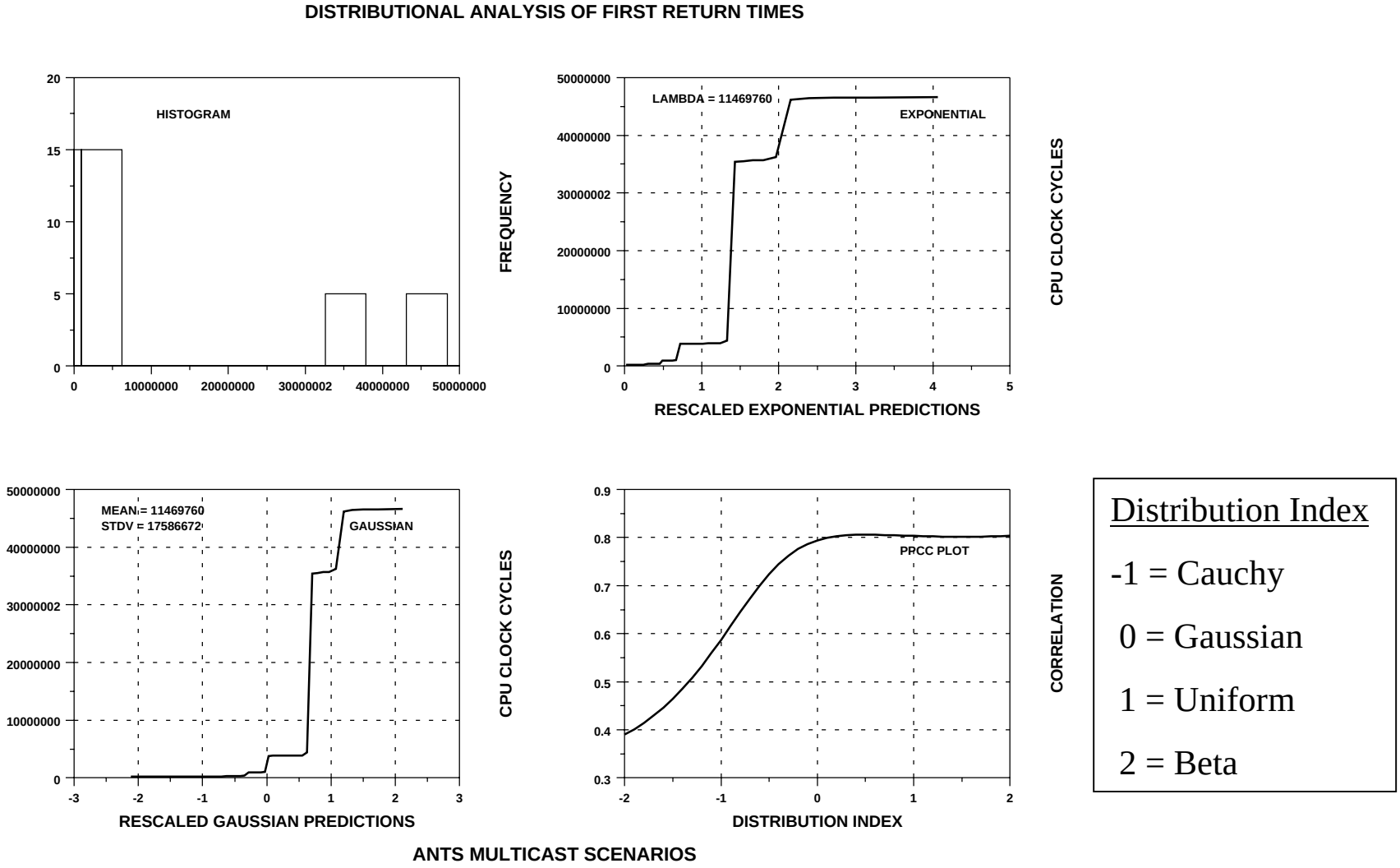
0 = Gaussian

1 = Uniform

2 = Beta

FROM STATE 102 TO STATE 004
ANTS PING : SCENARIO E (34 REP'S)

Proof-of-Concept:Results
Initial First-Return Times for ANTS Multicast



Distribution Index

-1 = Cauchy

0 = Gaussian

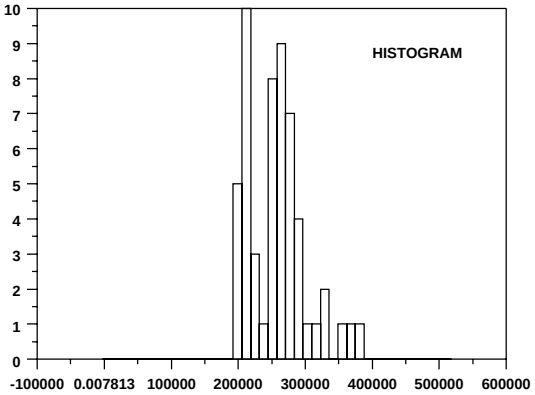
1 = Uniform

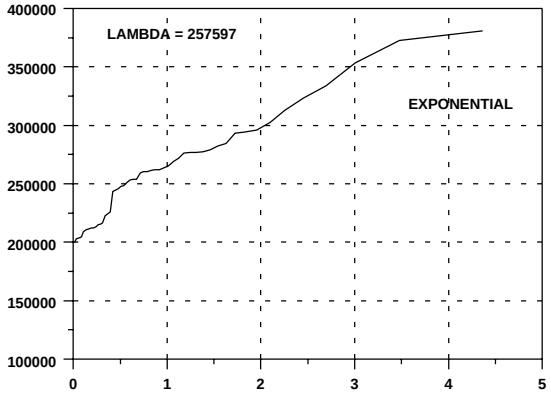
2 = Beta

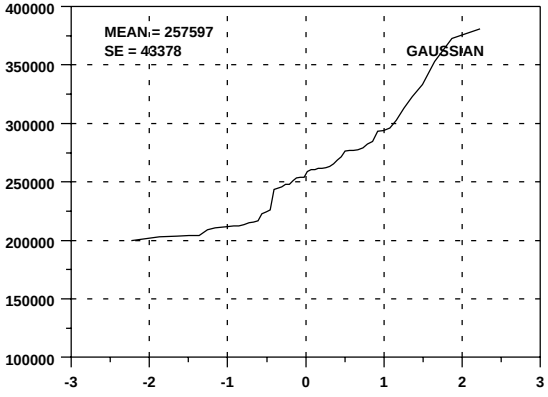
Proof-of-Concept:Results

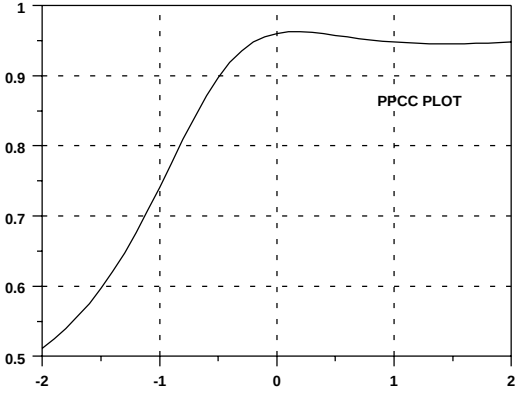
Transitions between a State Pair for ANTS Multicast

DISTRIBUTIONAL ANALYSIS OF DWELL+TRANSITION TIMES









Distribution Index

-1 = Cauchy

0 = Gaussian

1 = Uniform

2 = Beta

FROM STATE 106 TO STATE 106
ANTS MULTICAST : SCENARIO MA

Potential Benefits of Success

- Successful results will enable resource management systems on heterogeneous Active Nodes to address CPU time in addition to bandwidth and memory; nodes can enforce CPU usage contracts.

- Successful results will open new research possibilities in resource management for Active Networks.
  - Admission control decisions based on CPU, bandwidth, and memory requirements.
  - Find paths with sufficient CPU availability, while also meeting throughput, delay, and jitter requirements for an Active Application.
  - Query an Active Network with an Active Application's performance constraints and requirements for CPU time, memory, and bandwidth; sort through multiple path proposals with associated costs to select one.
  - Techniques might also apply to other mobile code systems intended for heterogeneous nodes.

Future Work: FY 00

Task 1: Develop and evaluate an Active Application (AA) model based on statistical analysis of AAs – let us know about yours!

Task 2: Design and develop a Self-Calibrating Active Node

- Calibration workload for EEs and for Node OS calls
- A self-calibration mechanism and related algorithms

Task 3: Design and implement an automated Active Application (AA) model generator

Task 4: Specify, design, and implement additional Node OS calls required to support calibration

Future Work: FY 01

Task 5: Prototype and evaluate our components as a system: across multiple Active Nodes, EEs, AAs, Node operating systems.

Task 6: Update prototype based on results from the evaluation.

Task 7: Integrate prototype with an Active Network resource manager.

Task 8: Demonstrate the ability to enforce CPU resource usage policy.